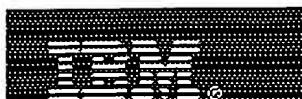
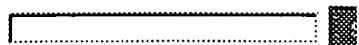


3/25/04



United States



Home | Products & services | Support & downloads | My account

Select a country

## Search results

Advanced search

Tips

Related links:

Technical support  
search

Downloads and drivers  
search

Employee directory

Search within:  United States  Worldwide

**polymorphic inline cache multithread**



[Advanced search](#) [Tips](#)

[polymorphic inline cache multithread](#)

**Limit your search by  
category:**

[Select One](#)



[1 - 10 of 23,614 results](#) | [Next >](#)

1. [\*\*RC23143.pdf\*\*](#)

Time compiler, **inline** caches to optimize **polymorphic** dispatch, and ...  
highlights include **polymorphic inline** caches, on ... size code **cache**  
then it could [More pages like this](#)

URL: <http://www.research.ibm.com/people/d/dgrose/papers/RC23143.pdf>

2. [\*\*TR-UW-CSE-96-06-02.pdf\*\*](#)

opportunity cost due to preventing **inline** expansion and post-inlining ...  
of statically-bound calls or **inlined** code. • Dynamic profile  
information [More pages like this](#)

URL: <http://www.research.ibm.com/people/d/dgrose/papers/TR-UW-CSE-96-06-02.pdf>

3. [\*\*pldi95.pdf\*\*](#)

optimizations such as **inline** expansion. One technique ... be statically-  
bound and **inlined**, the closure argument ... implemented the  
algorithm, use **polymorphic inline** caches [Hölzle et al  
URL: <http://www.research.ibm.com/people/d/dgrose/papers/pldi95.pdf>

4. [\*\*pepm94.pdf\*\*](#)

and potentially **inlined**. This can greatly ... implementation uses  
**polymorphic inline** caches (PICs ... method lookup **cache**, mapping  
argument

URL: <http://www.research.ibm.com/people/d/dgrose/papers/pepm94.pdf>

5. [\*\*oopsia96.pdf\*\*](#)

statically-bound or **inlined** version of the tar ... optimizations cross-  
module **inline** expansion Program Database ... one based  
on **polymorphic inline** caches (PICs) [Hölzle [More pages like this](#)

URL: <http://www.research.ibm.com/people/d/dgrose/papers/oopsia96.pdf>

6. [\*\*oopsia95.pdf\*\*](#)

some code really is **polymorphic**, manipulating objects of ... Even when  
code is **polymorphic**, it may be that one ... Self and Cecil, use  
**polymorphic inline** caches (PICs) [Hölzle

URL: <http://www.research.ibm.com/people/d/dgrose/papers/oopsia95.pdf>

7. [\*\*thesis.dvi\*\*](#)

by bringing data into the **cache** prior to a program's use ... 13 2.2.1.2  
**Multithreading** ... 107 5.2.2 **Cach** Statistics [More pages like this](#)

URL: <http://www.research.ibm.com/people/b/brendon/papers/thesis.dvi>

8. [\*\*WebSphere Application Server Version 5.1.x Information Center\*\*](#)

response none aatl0092w the bean **cache** must be configured to  
activate ... activitysession unless the bean **cache** is configured to

activate at ... user response configure the bean **cache** to activate at activitysession [More pages like this](#)  
URL: <http://www.ibm.com/software/info/testinfo.jsp?uid=IC0000...>

9. [\*\*Patents\*\*](#)  
thread switch in a **multithreaded** processor W. A ... data bus for sourcing **cache** memory data within ... Winterfield Multiprocessor **cache** coherence directed by ... fetch request in a **multithread** processor T. J  
URL: <http://researchweb.watson.ibm.com/journal/rd/452/patents...>
10. [\*\*Overview of the IBM Java Just-in-Time Compiler\*\*](#)  
rather than **polymorphic**. We **inline** the target ... testing and **inlined** the fast ... value of the **cache** that holds ... succeeds, the **cache** value is ... tests are **inlined** in the fast [More pages like this](#)  
URL: <http://researchweb.watson.ibm.com/journal/sj/391/suganum...>

[1 - 10 of 23,614 results](#) | [Next >](#)

[About IBM](#) | [Privacy](#) | [Terms of use](#) | [Contact](#)



> home > about > feedback > login

US Patent & Trademark Office



Try the *new* Portal design

Give us your opinion after using it.

## Search Results

Search Results for: [(polymorphic inline cache)]

Found 36 of 127,944 searched.

### Search within Results

> Advanced Search

> Search Help/Tips

Sort by: Title Publication Publication Date Score Binder

Results 1 - 20 of 36 short listing

Prev Page 1 2   
 Next Page

1 Reducing virtual call overheads in a Java VM just-in-time compiler 92%

Junpyo Lee , Byung-Sun Yang , Suhyun Kim , Kemal Ebcioğlu , Erik Altman , Seungil Lee , Yoo C. Chung , Heungbok Lee , Je Hyung Lee , Soo-Mook Moon

ACM SIGARCH Computer Architecture News March 2000

Volume 28 Issue 1

Java, an object-oriented language, uses *virtual methods* to support the extension and reuse of classes. Unfortunately, virtual method calls affect performance and thus require an efficient implementation, especially when just-in-time (JIT) compilation is done. *Inline caches* and *type feedback* are solutions used by compilers for dynamically-typed object-oriented languages such as SELF [1, 2, 3], where virtual call overheads are much more critical to performance than in Java. Wi ...

2 Efficient multiple and predicated dispatching 87%

Craig Chambers , Weimin Chen

ACM SIGPLAN Notices , Proceedings of the 14th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications October 1999

Volume 34 Issue 10

The speed of message dispatching is an important issue in the overall performance of object-oriented programs. We have developed an algorithm for constructing efficient dispatch functions that combines novel algorithms for efficient single dispatching, multiple dispatching, and predicate dispatching. Our algorithm first reduces methods written in the general predicate dispatching model (which generalizes single dispatching, multiple dispatching, predicate classes and classifiers, and patter ...

3 A study of devirtualization techniques for a Java Just-In-Time compiler 85%

Kazuaki Ishizaki , Motohiro Kawahito , Toshiaki Yasue , Hideaki Komatsu , Toshio Nakatani

ACM SIGPLAN Notices , Proceedings of the 15th ACM SIGPLAN conference on

## Object-oriented programming, systems, languages, and applications October 2000

### Volume 35 Issue 10

Many devirtualization techniques have been proposed to reduce the runtime overhead of dynamic method calls for various object-oriented languages, however, most of them are less effective or cannot be applied for Java in a straightforward manner. This is partly because Java is a statically-typed language and thus transforming a dynamic call to a static one does not make a tangible performance gain (owing to the low overhead of accessing the method table) unless it is inlined, and partly because t ...

**4** Reconciling responsiveness with performance in pure object-oriented 84%



Urs Hözle , David Ungar

**ACM Transactions on Programming Languages and Systems (TOPLAS)** July 1996

Volume 18 Issue 4

Dynamically dispatched calls often limit the performance of object-oriented programs, since object-oriented programming encourages factoring code into small, reusable units, thereby increasing the frequency of these expensive operations. Frequent calls not only slow down execution with the dispatch overhead per se, but more importantly they hinder optimization by limiting the range and effectiveness of standard global optimizations. In particular, dynamically dispatched calls prevent stand ...

**5** Rapid profiling via stratified sampling 82%



S. Subramanya Sastry , Rastislav Bodík , James E. Smith

**ACM SIGARCH Computer Architecture News , Proceedings of the 28th annual international symposium on Computer architecture** May 2001

Volume 29 Issue 2

*Sophisticated binary translators and dynamic optimizers demand a program profiler with low overhead, high accuracy, and the ability to collect a variety of profile types. A profiling scheme that achieves these goals is proposed. Conceptually, the hardware compresses a stream of profile data by counting identical events; the compressed profile data is passed to software for analysis. Compressing the high-bandwidth event stream greatly reduces software overhead. Because optimizations can take ...*

**6** Continuous program optimization: A case study 82%



Thomas Kistler , Michael Franz

**ACM Transactions on Programming Languages and Systems (TOPLAS)** July 2003

Volume 25 Issue 4

Much of the software in everyday operation is not making optimal use of the hardware on which it actually runs. Among the reasons for this discrepancy are hardware/software mismatches, modularization overheads introduced by software engineering considerations, and the inability of systems to adapt to users' behaviors. A solution to these problems is to delay code generation until load time. This is the earliest point at which a piece of software can be fine-tuned to the actual capabilities of the ...

**7** Design, implementation, and evaluation of optimizations in a just-in-time compiler 82%



Kazuaki Ishizaki , Motohiro Kawahito , Toshiaki Yasue , Mikio Takeuchi , Takeshi Ogasawara , Toshio Suganuma , Tamiya Onodera , Hideaki Komatsu , Toshio Nakatani

**Proceedings of the ACM 1999 conference on Java Grande** June 1999

**8** Vortex: an optimizing compiler for object-oriented languages 82%  
 Jeffrey Dean , Greg DeFouw , David Grove , Vassily Litvinov , Craig Chambers  
**ACM SIGPLAN Notices , Proceedings of the 11th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications** October 1996

Volume 31 Issue 10

Previously, techniques such as class hierarchy analysis and profile-guided receiver class prediction have been demonstrated to greatly improve the performance of applications written in pure object-oriented languages, but the degree to which these results are transferable to applications written in hybrid languages has been unclear. In part to answer this question, we have developed the Vortex compiler infrastructure, a language-independent optimizing compiler for object-oriented languages, with ...

**9** Minimizing row displacement dispatch tables 82%  
 Karel Driesen , Urs Hözle  
**ACM SIGPLAN Notices , Proceedings of the tenth annual conference on Object-oriented programming systems, languages, and applications** October 1995

Volume 30 Issue 10

Row displacement dispatch tables implement message dispatching for dynamically-typed languages with a run time overhead of one memory indirection plus an equality test. The technique is similar to virtual function table lookup, which is, however, restricted to statically typed languages like C++. We show how to reduce the space requirements of dispatch tables to approximately the same size as virtual function tables. The scheme is then generalized for multiple inheritance. Experiments on a numbe ...

**10** Type feedback vs. concrete type inference: a comparison of optimization 82%  
 techniques for object-oriented languages  
Ole Agesen , Urs Hözle  
**ACM SIGPLAN Notices , Proceedings of the tenth annual conference on Object-oriented programming systems, languages, and applications** October 1995

Volume 30 Issue 10

Two promising optimization techniques for object-oriented languages are type feedback (profile-based receiver class prediction) and concrete type inference (static analysis). We directly compare the two techniques, evaluating their effectiveness on a suite of 23 SELF programs while keeping other factors constant. Our results show that both systems inline over 95% of all sends and deliver similar overall performance with one exception: SELF's automatic coercion of machine integer ...

**11** Optimizing dynamically-dispatched calls with run-time type feedback 82%  
 Urs Hözle , David Ungar  
**ACM SIGPLAN Notices , Proceedings of the ACM SIGPLAN 1994 conference on Programming language design and implementation** June 1994

Volume 29 Issue 6

**12** A third-generation SELF implementation: reconciling responsiveness 82%  
 with performance  
Urs Hözle , David Ungar  
**ACM SIGPLAN Notices , Proceedings of the ninth annual conference on Object-oriented programming systems, language, and applications** October 1994

Volume 29 Issue 10

Programming systems should be both responsive (to support rapid development) and efficient (to complete computations quickly). Pure object-oriented languages are harder to implement efficiently since they need optimization to achieve good performance. Unfortunately, optimization conflicts with interactive responsiveness because it tends to produce long compilation pauses, leading to unresponsive programming environments. Therefore, to achieve good responsiveness, existing exploratory progra ...

**13 Incremental algorithms for dispatching in dynamically typed languages** 80%

 Yoav Zibin , Joseph (Yossi) Gil

**ACM SIGPLAN Notices , Proceedings of the 30th ACM SIGPLAN-SIGACT symposium on Principles of programming languages** January 2003

Volume 38 Issue 1

A fundamental problem in the implementation of object-oriented languages is that of a frugal *dispatching data structure*, i.e., support for quick response to dispatching queries combined with compact representation of the type hierarchy and the method families. Previous theoretical algorithms tend to be impractical due to their complexity and large hidden constant. In contrast, successful practical heuristics, including Vitek and Horspool's *compact dispatch tables* (CT) [16] designed ...

**14 Fast algorithm for creating space efficient dispatching tables with** 80%

 application to multi-dispatching

Yoav Zibin , Joseph Yossi Gil

**ACM SIGPLAN Notices , Proceedings of the 17th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications** November 2002

Volume 37 Issue 11

The dispatching problem can be solved very efficiently in the single-inheritance~(SI) setting. In this paper we show how to extend one such solution to the multiple-inheritance~(MI) setting. This generalization comes with an increase to the space requirement by a small factor of  $\kappa$ . This factor can be thought of as a metric of the complexity of the topology of the inheritance hierarchy. On a data set of~35 hierarchies totaling some~64 thousand types, our dispatching data structure, based on a ...

**15 A study of exception handling and its dynamic optimization in Java** 80%

 Takeshi Ogasawara , Hideaki Komatsu , Toshio Nakatani

**ACM SIGPLAN Notices , Proceedings of the 16th ACM SIGPLAN conference on Object oriented programming, systems, languages, and applications** October 2001

Volume 36 Issue 11

Optimizing exception handling is critical for programs that frequently throw exceptions. We observed that there are many such exception-intensive programs in various categories of Java programs. There are two commonly used exception handling techniques, stack unwinding optimizes the normal path, while stack cutting optimizes the exception handling path. However, there has been no single exception handling technique to optimize both paths.

**16 Efficient message dispatch in object-oriented systems** 80%

 Mayur Naik , Rajeev Kumar

**ACM SIGPLAN Notices** March 2000

Volume 35 Issue 3

Single dispatch involves performing at run-time a multi-way switch over the possible

classes of the receiver. Most object-oriented systems implement this switch as an array lookup using a table-based technique or as a binary search using a tree-based technique. However, each of these is the best choice only under a particular circumstance; neither outperforms the other under all circumstances. In this paper, we present a time and space efficient implementation of the switch that employs the tabl ...

**17 An automatic object inlining optimization and its evaluation** 80%

 Julian Dolby , Andrew Chien

**ACM SIGPLAN Notices , Proceedings of the ACM SIGPLAN 2000 conference on Programming language design and implementation** May 2000

Volume 35 Issue 5

Automatic object inlining [19, 20] transforms heap data structures by fusing parent and child objects together. It can improve runtime by reducing object allocation and pointer dereference costs. We report continuing work studying object inlining optimizations. In particular, we present a new semantic derivation of the correctness conditions for object inlining, and program analysis which extends our previous work. And we present an object inlining transformation, focusing ...

**18 Time and space efficient method-lookup for object-oriented programs** 80%

 S. Muthukrishnan , Martin Müller

**Proceedings of the seventh annual ACM-SIAM symposium on Discrete algorithms**

January 1996

**19 An evaluation of staged run-time optimizations in DyC** 80%

 Brian Grant , Matthai Philipose , Markus Mock , Craig Chambers , Susan J. Eggers

**ACM SIGPLAN Notices , Proceedings of the ACM SIGPLAN 1999 conference on Programming language design and implementation** May 1999

Volume 34 Issue 5

Previous selective dynamic compilation systems have demonstrated that dynamic compilation can achieve performance improvements at low cost on small kernels, but they have had difficulty scaling to larger programs. To overcome this limitation, we developed DyC, a selective dynamic compilation system that includes more sophisticated and flexible analyses and transformations. DyC is able to achieve good performance improvements on programs that are much larger and more complex than the kernels. We ...

**20 An evaluation of automatic object inline allocation techniques** 80%

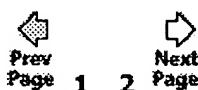
 Julian Dolby , Andrew A. Chien

**ACM SIGPLAN Notices , Proceedings of the 13th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications**

October 1998

Volume 33 Issue 10

Object-oriented languages such as Java and Smalltalk provide a uniform object reference model, allowing objects to be conveniently shared. If implemented directly, these uniform reference models can suffer in efficiency due to additional memory dereferences and memory management operations. Automatic *inline allocation* of child objects within parent objects can reduce overheads of heap-allocated pointer-referenced objects. We present three compiler analyses to identify inlinable fields by t ...



---

The ACM Portal is published by the Association for Computing Machinery. Copyright ?2004 ACM, Inc.



> home > about > feedback > login

US Patent & Trademark Office



Try the *new* Portal design

Give us your opinion after using it.

## Search Results

Search Results for: [multi-thread <AND>(((polymorphic inline cache ) ) ) ]  
Found 5 of 127,944 searched.

### Search within Results

> Advanced Search

> Search Help/Tips

Sort by: Title Publication Publication Date Score Binder

Results 1 - 5 of 5 short listing

1 A study of devirtualization techniques for a Java Just-In-Time compiler 82%

Kazuaki Ishizaki , Motohiro Kawahito , Toshiaki Yasue , Hideaki Komatsu , Toshio Nakatani  
**ACM SIGPLAN Notices , Proceedings of the 15th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications** October 2000 Volume 35 Issue 10

Many devirtualization techniques have been proposed to reduce the runtime overhead of dynamic method calls for various object-oriented languages, however, most of them are less effective or cannot be applied for Java in a straightforward manner. This is partly because Java is a statically-typed language and thus transforming a dynamic call to a static one does not make a tangible performance gain (owing to the low overhead of accessing the method table) unless it is inlined, and partly because t ...

2 Automatic program specialization for Java 77%

Ulrik P. Schultz , Julia L. Lawall , Charles Consel  
**ACM Transactions on Programming Languages and Systems (TOPLAS)** July 2003 Volume 25 Issue 4

The object-oriented style of programming facilitates program adaptation and enhances program genericness, but at the expense of efficiency. We demonstrate experimentally that state-of-the-art Java compilers fail to compensate for the use of object-oriented abstractions in the implementation of generic programs, and that program specialization can eliminate a significant portion of these overheads. We present an automatic program specializer for Java, illustrate its use through detailed case stud ...

3 Reducing virtual call overheads in a Java VM just-in-time compiler 77%

Junpyo Lee , Byung-Sun Yang , Suhyun Kim , Kemal Ebcioğlu , Erik Altman , Seungil Lee , Yoo C. Chung , Heungbok Lee , Je Hyung Lee , Soo-Mook Moon  
**ACM SIGARCH Computer Architecture News** March 2000 Volume 28 Issue 1

Java, an object-oriented language, uses *virtual methods* to support the extension and reuse of classes. Unfortunately, virtual method calls affect performance and thus require an efficient implementation, especially when just-in-time (JIT) compilation is

done. *Inline caches and type feedback* are solutions used by compilers for dynamically-typed object-oriented languages such as SELF [1, 2, 3], where virtual call overheads are much more critical to performance than in Java. Wi ...

4 Design, implementation, and evaluation of optimizations in a just-in-time compiler 77%

Kazuaki Ishizaki , Motohiro Kawahito , Toshiaki Yasue , Mikio Takeuchi , Takeshi Ogasawara , Toshio Suganuma , Tamiya Onodera , Hideaki Komatsu , Toshio Nakatani  
**Proceedings of the ACM 1999 conference on Java Grande** June 1999

5 An evaluation of automatic object inline allocation techniques 77%

Julian Dolby , Andrew A. Chien  
**ACM SIGPLAN Notices , Proceedings of the 13th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications** October 1998  
Volume 33 Issue 10

Object-oriented languages such as Java and Smalltalk provide a uniform object reference model, allowing objects to be conveniently shared. If implemented directly, these uniform reference models can suffer in efficiency due to additional memory dereferences and memory management operations. Automatic *inline allocation* of child objects within parent objects can reduce overheads of heap-allocated pointer-referenced objects. We present three compiler analyses to identify inlinable fields by t ...

---

**Results 1 - 5 of 5      short listing**

---

The ACM Portal is published by the Association for Computing Machinery. Copyright ?2004 ACM, Inc.

[Subscribe \(Full Service\)](#) [Register \(Limited Service, Free\)](#) [Login](#)Search:  The ACM Digital Library  The Guide 

THE ACM DIGITAL LIBRARY

[Feedback](#) [Report a problem](#) [Satisfaction survey](#)

## An evaluation of automatic object inline allocation techniques

**Full text**  [Pdf \(2.26 MB\)](#)**Source** Conference on Object Oriented Programming Systems Languages and Applications [archive](#)  
Proceedings of the 13th ACM SIGPLAN conference on Object-oriented programming,  
systems, languages, and applications [table of contents](#)  
Vancouver, British Columbia, Canada  
Pages: 1 - 20  
Year of Publication: 1998  
ISSN:0362-1340  
[Also published in ...](#)**Authors** Julian Dolby  
[Andrew A. Chien](#)**Sp nsor** SIGPLAN: ACM Special Interest Group on Programming Languages**Publisher** ACM Press New York, NY, USA**Additional Information:** [abstract](#) [references](#) [citations](#) [index terms](#) [collaborative colleagues](#) [peer to peer](#)**Tools and Actions:** [Discussions](#) [Find similar Articles](#) [Review this Article](#)  
[Save this Article to a Binder](#) [Display in BibTex Format](#)**DOI Bookmark:** Use this link to bookmark this Article: <http://doi.acm.org/10.1145/286936.286943>  
[What is a DOI?](#)

### ↑ ABSTRACT

Object-oriented languages such as Java and Smalltalk provide a uniform object reference model, allowing objects to be conveniently shared. If implemented directly, these uniform reference models can suffer in efficiency due to additional memory dereferences and memory management operations. Automatic *inline allocation* of child objects within parent objects can reduce overheads of heap-allocated pointer-referenced objects. We present three compiler analyses to identify inlinable fields by tracking accesses to heap objects. These analyses span a range from local data flow to adaptive whole-program, flow-sensitive inter-procedural analysis. We measure their cost and effectiveness on a suite of moderate-sized C++ programs (up to 30,000 lines including libraries). We show that aggressive interprocedural analysis is required to enable object inlining, and our adaptive inter-procedural analysis [23] computes precise information efficiently. Object inlining eliminates typically 40% of object accesses and allocations (improving performance up to 50%). Furthermore,

### ↑ REFERENCES

Note: OCR errors may be found in this Reference List extracted from the full text article. ACM has opted to expose the complete List rather than only correct and linked references.

<sup>1</sup> O. Agesen, J. Palsberg, and M. Schwartzbach. Type inference of SELF: Analysis of objects with dynamic and multiple inheritance. In Proceedings of ECOOP '93, 1993.

[Subscribe \(Full Service\)](#) [Register \(Limited Service, Free\)](#) [Login](#)Search:  The ACM Digital Library  The Guide

THE ACM DIGITAL LIBRARY

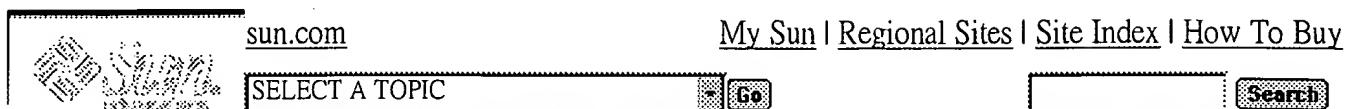
[Feedback](#) [Report a problem](#) [Satisfaction survey](#)

## A study of devirtualization techniques for a Java Just-In-Time compiler

**Full text** [Pdf \(226 KB\)](#)**Source** Conference on Object Oriented Programming Systems Languages and Applications [archive](#)  
Proceedings of the 15th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications [table of contents](#)  
Minneapolis, Minnesota, United States  
Pages: 294 - 310  
Year of Publication: 2000  
ISBN: 1-58113-200-X  
[Also published in ...](#)**Authors** Kazuaki Ishizaki  
[Motohiro Kawahito](#)  
[Toshiaki Yasue](#)  
[Hideaki Komatsu](#)  
[Toshio Nakatani](#)**Sp nsor** SIGPLAN: ACM Special Interest Group on Programming Languages**Publisher** ACM Press New York, NY, USA**Additional Information:** [abstract](#) [references](#) [citations](#) [index terms](#) [collaborative colleagues](#) [peer to peer](#)**To Is and Actions:** [Discussions](#) [Find similar Articles](#) [Review this Article](#)  
[Save this Article to a Binder](#) [Display in BibTeX Format](#)**DOI Bookmark:** Use this link to bookmark this Article: <http://doi.acm.org/10.1145/353171.353191>  
[What is a DOI?](#)

### ABSTRACT

Many devirtualization techniques have been proposed to reduce the runtime overhead of dynamic method calls for various object-oriented languages, however, most of them are less effective or cannot be applied for Java in a straightforward manner. This is partly because Java is a statically-typed language and thus transforming a dynamic call to a static one does not make a tangible performance gain (owing to the low overhead of accessing the method table) unless it is inlined, and partly because the dynamic class loading feature of Java prohibits the whole program analysis and optimizations from being applied. We propose a new technique called *direct devirtualization with the code patching mechanism*. For a given dynamic call site, our compiler first determines whether the call can be devirtualized, by analyzing the current class hierarchy. When the call is devirtualizable and the target method is suitably sized, the compiler generates the inlined code of the method, together with the backup code of making the dynamic call. Only the inlined code is actually executed until our assumption about the devirtualization becomes invalidated, at which time the compiler performs code patching to make the backup code executed subsequently. Since the new technique prevents some code motions across the merge point between the inlined code and the backup code, we have furthermore implemented recently-known analysis techniques, such as type analysis and preexistence analysis, which allow the backup code to be completely eliminated. We made various experiments using 16 real programs to understand the effectiveness and characteristics of the devirtualization techniques in our Java Just-In-Time (JIT) compiler. In summary, we reduced the number of dynamic calls by ranging from 8.9% to 97.3% (the average of 40.2%), and we improved



[Home](#) > [Research](#) > [Projects](#) > [Previous Projects](#) > [Self](#)

› Collaborative Computing Group

› Self

- [Self 4.1](#)

- [Self 4.0](#)

- [Papers](#)

- [Tutorial](#)

› Service Location Protocol

# Optimizing Dynamically-Dispatched Calls with Run-Time Type Feedback

Urs Hözle and David Ungar

**Abstract:** Object-oriented programs are difficult to optimize because they execute many dynamically-dispatched calls. These calls cannot easily be eliminated because the compiler does not know which callee will be invoked at runtime. We have developed a simple technique that feeds back type information from the runtime system to the compiler. With this type feedback, the compiler can inline any dynamically-dispatched call. Our compiler drastically reduces the call frequency of a suite of large SELF applications (by a factor of 3.6) and improves performance by a factor of 1.7. We believe that type feedback could significantly reduce call frequencies and improve performance for most other object-oriented languages (statically-typed or not) as well as for languages with type-dependent operations such as generic arithmetic.

*Proceedings of the ACM SIGPLAN'94 Conference on Programming Language Design and Implementation.*

To get the PostScript file, click [here](#).

[Back to bibliography](#)

Self : [Self 4.1](#) | [Self 4.0](#) | [Papers](#) | [Tutorial](#)

Copyright 1994-2003 Sun Microsystems, Inc., 901 San Antonio Road, Palo Alto, CA 94303 USA. All rights reserved.

[Terms of Use](#). [Privacy Policy](#). [Feedback](#)



[Subscribe \(Full Service\)](#) [Register \(Limited Service, Free\)](#) [Login](#)

Search:  The ACM Digital Library  The Guide

[SEARCH](#)

THE ACM DIGITAL LIBRARY

[Feedback](#) [Report a problem](#) [Satisfaction survey](#)

## The design and evaluation of a high performance SMALLTALK system

[Purchase a copy.](#)

**Source** Year of Publication: 1986  
Order Number: UMI order no. GAX86-24972

**Author** [David Michael Ungar](#)

**Publisher** University of California at Berkeley Berkeley, CA, USA

**Additional Information:** [citations](#) [index terms](#)

**Tools and Actions:** [Discussions](#) [Find similar Doctoral Theses](#) [Review this Doctoral Thesis](#)  
[Save this Doctoral Thesis to a Binder](#) [Display in BibTeX Format](#)

### ↑ CITINGS 12

[Douglas Johnson, The case for a read barrier, ACM SIGARCH Computer Architecture News, v.19 n.2, p.279-287, Apr. 1991](#)

[Benjamin Goldberg, Tag-free garbage collection for strongly typed programming languages, ACM SIGPLAN Notices, v.26 n.6, p.165-176, June 1991](#)

[Douglas Johnson, Trap architectures for Lisp systems, Proceedings of the 1990 ACM conference on LISP and functional programming, p.79-86, June 27-29, 1990, Nice, France](#)

[A. Dain Samples , David Ungar , Paul Hilfinger, SOAR: Smalltalk without bytecodes, Conference proceedings on Object-oriented programming systems, languages and applications, p.107-118, September 29-October 02, 1986, Portland, Oregon, United States](#)

[Peter Steenkiste , John Hennessy, LISP on a reduced-instruction-set-processor, Proceedings of the 1986 ACM conference on LISP and functional programming, p.192-201, August 1986, Cambridge, Massachusetts, United States](#)

[W. H. Harrison , P. F. Sweeney , J. J. Shilling, Good news, bad news: experience building software development environment using the object-oriented paradigm, ACM SIGPLAN Notices, v.24 n.10, p.85-94, Oct. 1989](#)

[Ravi Sharma , Mary Lou Soffa, Parallel generational garbage collection, ACM SIGPLAN Notices, v.26 n.11, p.16-32, Nov. 1991](#)

[Benjamin Zorn, Comparing mark-and sweep and stop-and-copy garbage collection, Proceedings of the 1990 ACM conference on LISP and functional programming, p.87-98, June 27-29, 1990, Nice, France](#)

*Call for Participation*

The 4th Annual Workshop on  
Workshop on Interaction between Compilers and Computer Architectures  
(INTERACT-4)

in conjunction with the

6th International Symposium on  
High-Performance Computer Architecture (HPCA-6)

Toulouse - France, January 9, 2000

Organizers: Gyungho Lee, Iowa State University and  
Pen-Chung Yew, University of Minnesota

---

Advance Program

SESSION I (10:00 - 11:30 a.m.)

Load-Store Optimization for Software Pipelineing  
Min Dai, Christine Eisenbeis, and Sid-Ahmed-Ali Touati

Automatic Memory Layout Transformations to Optimize Spatial Locality in Parameterized Loop Nests  
Philippe Clauss and Benoit Meister

Limits of Task-based Parallelism in Irregular Applications  
Barbara Krieseck, Dean Tullsen, and Brad Calder

- Lunch Break -

SESSION II (1:30 - 2:30 p.m.)

Reducing Virtual Call Overheads in a Java VM Just-In-Time Compiler  
Junpyo Lee, Byung-Sun Yang, Suhyun Kim, SeungIl Lee, Yoo C. Chung, Heungbok Lee, Je Hyung Lee, Soo-Mook Moon, Kemal Ebcioglu, and Erik Altman

Applying Predication to Efficiently Handle Runtime Class Testing  
Chris Sandler, Sandeep K. S. Gupta, and Rohit Bhatia

- Coffee Break -

SESSION III (3:00 - 4:30 p.m.)

Optimizing Cache Miss Equation Polyhedra

Nerina Bermudo, Xavier Vera, Antonio Gonzalez, and Josep Llosa

A Combined Compiler and Architecture Technique to Control Multithreaded Execution and Branches,  
and Loop Iterations

A. Unger, E. Zehendner, and T. Ungerer

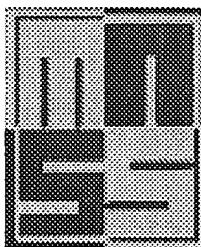
Using Cache Line Coloring to Perform Aggressive Procedure Inlining

Hakan Aydin and David Kaeli

---

Updated on December 26, 1999

# Byung-Sun Yang's Homepage



- [Introduction](#)
- [Research](#)
- [Publications](#)
- [Postponi](#)

## Publications

- Efficient Java Exception Handling in Just-in-Time Compilation  
*SeungIl Lee, Byung-Sun Yang, Suhyun Kim, Seongbae Park, Soo-Mook Moon, Kemal Ebciooglu, and Erik Altman*  
To appear in the ACM 2000 Java Grande Conference, San Francisco, California, June 2000.
- Reducing Virtual Call Overheads in a Java VM Just-in-Time Compiler  
*Junpyo Lee, Byung-Sun Yang, Suhyun Kim, SeungIl Lee, Yoo C. Chung, Heungbok Lee, Je Hyung Lee, Soo-Mook Moon, Kemal Ebciooglu, and Erik Altman*  
In Proceedings of 1999 Workshop on Interaction between Compilers and Computer Architectures, Toulouse, France, January 2000.
- LaTTe: A Java VM Just-in-Time Compiler with Fast and Efficient Register Allocation  
*Byung-Sun Yang, Soo-Mook Moon, Seongbae Park, Junpyo Lee, SeungIl Lee, Jinpyo Park, Yoo C. Chung, Suhyun Kim, Kemal Ebciooglu, and Erik Altman*  
In Proceedings of the 1999 International Conference on Parallel Architectures and Compilation Techniques (PACT'99), New Port Beach, California, October 1999.
- On-Demand Translation of Java Exception Handlers in the LaTTe JVM Just-in-Time Compiler  
*SeungIl Lee, Byung-Sun Yang, Suhyun Kim, Seongbae Park, Soo-Mook Moon, Kemal Ebciooglu, and Erik Altman*  
In Proceedings of the 1999 Workshop on Binary Translation (Binary99), New Port Beach, California, October 1999.
- Lightweight Monitor in Java Virtual Machine  
*Byung-Sun Yang, Junpyo Lee, Jinpyo Park, Soo-Mook Moon, and Kemal Ebciooglu*  
In Proceedings of the Third Workshop on Interaction between Compilers and Computer Architectures (INTERACT-3), San Jose, California, October 7, 1998.
- Effect of Memory Disambiguation for ILP Microprocessors (Korean)  
*HoeMok Chung, Byung-Sun Yang, Soo-Mook Moon*  
In Proceedings of The 25th KISS Fall Conference, October 1998.
- Design of a Java Virtual Machine using Enhanced JIT Compile Technique (Korean)  
*Byung-Sun Yang and Soo-Mook Moon*  
In Proceedings of The 24th KISS Fall Conference, October

1997.

{